{EPITECH}

POPEYE_

< BECOME A TRUE DOCKER SAILOR ! />

POPEYE

Docker is a software used to containerize application, which means, you can run the same application on different OS without installing the dependencies on your computer. It's a practice tool for group work development because it's lighter, faster and securer.

Also, containerization consist in packaging various things inside one simple standard object, easier to work with. It eases many manipulations (carrying, moving, stacking, ...) and increases tremendously the productivity.

Since you're at Epitech, how many times do you struggle to install dependencies ?

Then, you have to help your mates with his/her computer. (Especially MACBOOK!!!)

In this project, you will understand and master the power of containerization.

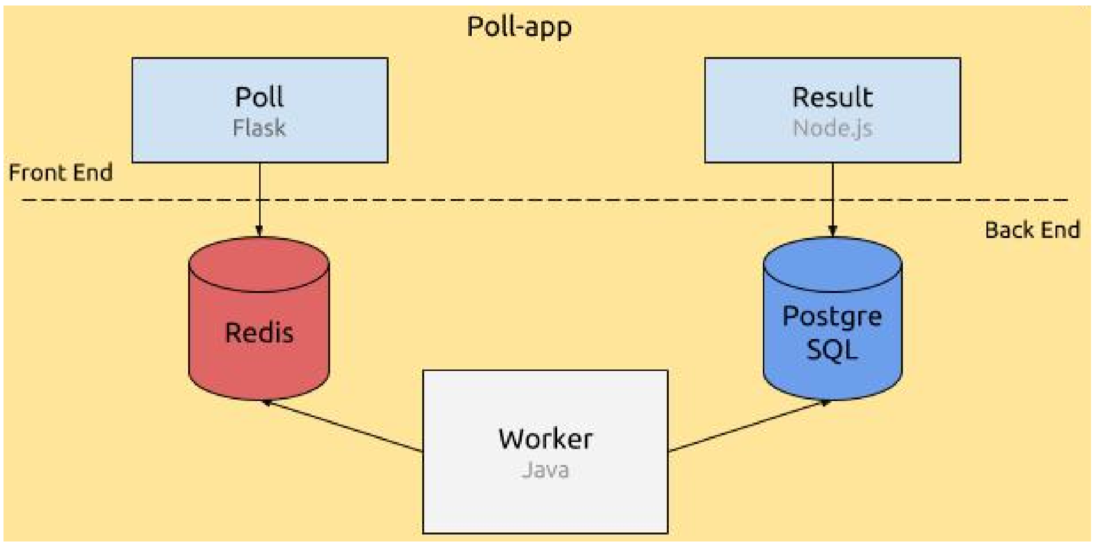
Like Popeye eating spinach to gain strength, you will get a buff for your next projects at Epitech with Docker and Compose.

General description

You will have to containerize and define the deployment of a web poll application.

There are five elements constituting the application:

- ✓ ***Poll***, a Flask Python web application that gathers votes and push them into a Redis queue.
- ✓ A **Redis queue**, which holds the votes sent by the Poll application, awaiting them to be consumed by the Worker.
- ✓ The ***Worker***, a Java application which consumes the votes being in the Redis queue, and stores them into a PostgreSQL database.
- ✓ A **PostgreSQL database**, which (persistently) stores the votes stored by the Worker.
- ✓ ***Result***, a Node.js web application that fetches the votes from the database and displays them.



Do not worry, you do not have to code them! Popeye is generous, the code of these applications is given to you alongside the project's subject.

In DevOps, it is especially important that you take the time to research and understand the technologies you are asked to work with, as you will need to understand how and by which way you can configure them as needed.

Docker images

You have to create 3 images, respecting the specifications described below.

Popeye does not like the `ENTRYPOINT` instruction, so you must not use it in this project.

It's part of your duty to find which versions the different elements of your infrastructure need.

Latest versions is a bad practice.
If a major update comes, will your project still be able to run?

Poll

- ✓ the image is based on an official Python image ;
- ✓ the app exposes and runs on port 80 ;

Result

- ✓ the image is based on an official Node.js Alpine image ;
- ✓ the app exposes and runs on port 80 ;

The `node_modules` folder must be excluded from the build context.

Worker

The image is built using a multi-stage build:

- ✓ First stage - compilation:
  - is based on `maven:3.9.6-eclipse-temurin-21-alpine` and is named `builder`.
  - is used to build and package the Worker application using:
    - `mvn dependency:resolve` from within the folder containing `pom.xml`;

    * then `mvn package` from within the folder containing the `src` folder.
  - generates a file in the `target` folder named `worker-jar-with-dependencies.jar`

✓ Second stage - run:

  - is based on `eclipse-temurin:21-jre-alpine` ;
  - is the one really running the worker using `java -jar worker-jar-with-dependencies.jar`.

Your Docker images must be as simple and lightweight as possible.

Connection details (such as addresses, ports, credentials, ...) are coded with environment variables. Learn how to use them.

NEVER push the env file!

Docker Compose

Now, you should have 3 Dockerfiles creating 3 isolated images.
It is now time to make them all work together using Docker Compose!

Create a `compose.yml` file that will be responsible for running and linking your containers.

Your Compose file should contain:

✓ **5 services**:

- `poll`:
  - ⋆ builds your `poll` image ;
  - ⋆ redirects port 5000 of the host to the port 80 of the container.
- `redis`:
  - ⋆ uses an existing official image of Redis ;
  - ⋆ opens port 6379.
- `worker`:
  - ⋆ builds your `worker` image.
- `db`:
  - ⋆ represents the database that will be used by the apps ;
  - ⋆ uses an existing official image of PostgreSQL;
  - ⋆ has its database schema created during container first start.
- `result`:
  - ⋆ builds your `result` image ;
  - ⋆ redirects port 5001 of the host to the port 80 of the container.

Databases must be launched before the services that use them, because these services *depend on* them.

✓ **3 networks**:

- `poll-tier`, `result-tier` and `back-tier`. It's your job to think about which services should be included in which network.

The schema of the project on top should help you.

✓ **1 named volume**:
  - db-data which allows the database's data to be persistent, if the container dies.

Do not add unnecessary data, such as extra volumes, extra networks, unnecessary inter-container dependencies, or unnecessary container commands/entrypoints.

Once your compose.yml is complete, you should be able to run all the services and observe the votes you submitted to the poll's webpage on result's webpage.

Your containers must restart automatically when they stop unexpectedly.

Technical formalities

Your project will be entirely evaluated with automated tests, by analyzing your configuration files (the different Dockerfiles and compose.yml).

In order to be correctly evaluated, your repository must at least contain the following files:

```
|-- compose.yml
|-- poll
|    `-- Dockerfile
|-- result
|    `-- Dockerfile
|-- schema.sql
`-- worker
     `-- Dockerfile
```

poll, result and worker are the directories containing the provided applications.

v 1.2

{EPITECH}